Option informatique

Présentation du sujet

Le sujet porte sur des systèmes de vote à effectuer lors d'une élection. La première partie traite du vote par préférence et évoque le théorème de McGarvey sur sa relation avec les matrices. La deuxième partie présente les vainqueurs de Condorcet, ainsi que la méthode de sélection d'un vainqueur de Schulze. La dernière partie présente un problème de décision (`Control-Alt-Add`) et propose de montrer sa NP-complétude.

Les candidats ont bien abordé l'ensemble du problème et les meilleurs d'entre eux ont traité toutes les questions.

Analyse globale des résultats

Le sujet a globalement été bien compris et le jury est globalement satisfait d'une majorité des candidats. Les questions de programmation sont plutôt bien traitées, mais comme chaque année, nous constatons des candidats qui ont une maitrise très superficielle de la syntaxe `Caml`, la mélangent avec celle de `Python`, et ne comprennent pas les différences fondamentales entre ces deux langages.

Certaines questions théoriques ont perturbé une partie des candidats de par leur composante mathématique forte. De même, la dernière partie sur la NP-complétude en a déstabilisé plus d'un.

Cette année, des consignes sans ambiguïté étaient présentées en début de sujet sur les fonctions informatique autorisées pour la composition. Certains candidats ont toutefois réécrit des fonctions déjà existantes dans les librairies autorisées, soit par ignorance de ces fonctions, soit car ils n'avaient pas lu correctement les consignes.

Malgré ces remarques, il reste beaucoup de candidats qui rendent des copies exemplaires, avec des justifications détaillées pour expliquer le rôle des fonctions auxiliaires qui interviennent et le fonctionnement global des algorithmes mis en place.

Commentaires sur les réponses apportées et conseils aux futurs candidats

De manière non exhaustive, nous citons des erreurs fréquentes rencontrées dans les copies :

— l'oubli que les variables dans les filtrages sont muettes et que l'usage du `when` est parfois nécessaire. Par exemple, `match x with y -> ...` doit souvent être remplacé par une syntaxe de la forme `match x with _ when x = y -> ...` ;

— l'utilisation maladroite des références, voire leur oubli total. Nous constatons une absence régulière du `!`, ou l'utilisation de `=` au lieu de `:=`. Nous rappelons qu'une liste est un objet non modifiable. Ainsi, le code suivant

```
let l = [] in
for i = 0 to n - 1 do
    l @ [i]
done
```

ne fait absolument rien : une nouvelle liste est créée à chaque passage dans la boucle, puis oubliée au passage suivant ;

— la confusion perpétuelle entre listes et tableaux, dans un sens comme dans l'autre, même en fin de deuxième année de classe préparatoire. Si `u` est une liste, `u.(i)` n'a aucun sens. De même, la commande `List.make` n'existe pas. Une lecture attentive de l'énoncé peut souvent permettre à ceux qui connaissent ces différences d'éviter de perdre des points.

Par ailleurs, les questions de complexité sont parfois mal justifiées, et en particulier, les candidats oublient que la complexité temporelle de `u @ v` n'est pas nécessairement la même que celle de `v @ u`. Nous rappelons qu'en utilisant la notation $\mathcal{O}$, il est largement préférable de conclure qu'une complexité est un $\mathcal{O}(n^2p)$ plutôt qu'un $\mathcal{O}(\frac{1}{2}n(n-1)p)$. Enfin, il est nécessaire de comprendre que la complexité d'un appel à `f (g x)` n'est pas le produit, mais la *somme* des complexités de `f` et `g`.

Aussi, les candidats qui écrivent plusieurs fonctions auxiliaires appelées `aux1`, `aux2`, etc. ne peuvent se le permettre que s'ils apportent des explications claires sur le rôle de ces fonctions et de leurs arguments. Les membres de jury ne sont pas (encore ?) des ordinateurs qui compilent automatiquement le code qu'ils lisent pour le comprendre.

Si la programmation récursive et la programmation itérative sont interchangeables, certaines fonctions sont plus facile à écrire et à lire lorsqu'elle le sont avec des boucles. Certains candidats ont mal écrit des fonctions récursives, pénibles à relire et avec des cas d'arrêts erronés, là où une double boucle `for` faisait tout aussi bien le travail. À ce sujet, nous rappelons que dans la syntaxe `for i = a to b do`, les bornes `a` et `b` sont toutes les deux atteintes.

Pour en terminer avec les remarques générales, il y a un réel effort de présentation pour la très grande majorité des copies. Cela rend d'autant plus visible les quelques copies rendues quasiment sous forme de brouillon (résultats soulignés ou encadrés sans règle, ratures omniprésentes, corrections écrites en petit et en travers au milieu des ratures, etc.). De tels candidats ont bien entendu été pénalisés. Par ailleurs, nous rappelons aux candidats que l'usage d'un français correct est de rigueur : respect de l'orthographe, des règles de grammaire et de conjugaison. Enfin, les abréviations et le style télégraphique sont à proscrire.

Il y a eu quelques incompréhensions sur la question **Q1** du sujet, car plusieurs candidats ont compris qu'il fallait écrire la fonction dans le cas de l'exemple qui était donné (la sous-partie en question s'appelle effectivement « Premier exemple »). À cette question, plusieurs candidats ont considéré que `u` était en fait une matrice et non une liste de listes. Le cas des coefficients diagonaux a souvent été omis à la question **Q4**, alors que la fonction `duel` ne traitait pas de manière particulière le cas d'égalité entre i et j. Nous avons souvent vu des preuves utilisant des bases d'espaces vectoriels pour la question **Q7**, mais rappelons que $\mathbb{Z}$ n'est pas un corps ! Plusieurs problèmes de complexité ont été rencontrés à la question **Q8**. Le jury rappelle que même si aucune complexité n'est imposée à une question, il est recommandé d'écrire une fonction avec la meilleure complexité possible tant que cela ne nécessite pas une réécriture complète de l'algorithme. Ainsi, on préfèrera `i :: l` à `l @ [i]` en termes de complexité.

La confusion entre listes et tableaux s'est à nouveau présentée à la question **Q11** où une part non négligeable a travaillé avec des tableaux de listes au lieu de matrices. Les questions **Q13**, **Q14** et **Q15** ont posé problème aux candidats qui n'ont pas réussi à s'adapter au fait que la notion de distance entre les sommets n'était pas la notion usuelle. Nous rappelons également que l'algorithme de Floyd-Warshall est au programme de deuxième année et ceux qui ont montré qu'ils connaissaient leurs cours ont été récompensés. La question **Q22** était difficile, mais les candidats qui ont correctement utilisé l'indication fournie par l'énoncé ont réussi à justifier correctement la transitivité.

Si un grand nombre de candidats ont indiqué qu'il existait 2^n distributions de vérité pour la question **Q24**, très peu ont justifié correctement qu'il était possible de les générer avec une complexité $\mathcal{O}(2^n)$. La justification que le test d'une distribution se faisait en $\mathcal{O}(m)$ (qui était le nombre de littéraux et non le nombre de clauses) était également rarement présente. Les candidats qui, à la question **Q25**, ont tracé le

graphe en ne représentant que les arcs strictement positifs ont été appréciés, en comparaison aux schémas illisibles qui ont parfois été rencontrés. Les questions **Q26**, **Q27** et **Q28** ont été peu traitées par les candidats.

Conclusion

Malgré des conditions de travail inhabituelles et difficiles pour cette fin de deuxième année, le jury est satisfait de voir qu'un grand nombre de candidats ont un niveau tout à fait satisfaisant en informatique, tant pour la rédaction des questions théoriques que l'écriture de programmes clairs et compréhensibles.

Nous sommes tout à fait conscients qu'il est difficile d'écrire du code sur papier, mais le jury recommande toutefois de s'entrainer sur ordinateur au cours de l'année pour acquérir les bons réflexes.